이동 객체의 위치 추적을 위한 KDB-트리 기반의 시공간 색인구조☆

Spatio-Temporal Index Structure based on KDB-Tree for Tracking Positions of Moving Objects

서 동 민* 복 경 수** 유 재 수*** 이 병 엽****
Dong-Min Seo Kyoung-Soo Bok Jae Soo Yoo Byoung-Yup Lee

요 약

최근 위치기반 기술의 급속한 발전으로 인하여 이동 객체를 효율적으로 관리하기 위한 색인 구조의 필요성이 증가하고 있다. 하지만, 기존에 제안된 색인 구조들은 이동 객체의 계속되는 위치 이동으로 인해 색인의 변경이 발생하고 색인의 빈번한 변경으로 전체적인 색인의 성능이 저하된다. 본 논문에서는 KDB-트리를 기반으로 하는 시공간 색인 구조인 TPKDB-트리를 제안한다. 제안하는 색인 구조는 갱신 비용을 최소화하여 이동 객체 검색의 효율성을 증가시키고 이동 객체를 선형함수로 표현함으로서 불필요한 갱신을 줄이는 방법을 제안한다. 그리고 노드 내에 포함되어 있는 이동 객체의 변화를 시간에 대한 파라미터로 유지함으로서 효율적으로 이동 객체의 미래 위치 검색을 지원한다. 또한, 공간활용도를 최대화하기 위한 새로운 갱신 및 분할 기법을 제안한다. 제안하는 색인 구조의 우수성을 입증하기 위해 다양한 실험을 통해 성능 평가를 수행한다.

Abstract

Recently, the needs of index structure which manages moving objects efficiently have been increased because of the rapid development of location-based techniques. Existing index structures frequently need updates because moving objects change continuatively their positions. That caused entire performance loss of the index structures. In this paper, we propose a new index structure called the TPKDB-tree that is a spatio-temporal index structure based on KDB-tree. Our technique optimizes update costs and reduces a search time for moving objects and reduces unnecessary updates by expressing moving objects as linear functions. Thus, the TPKDB-tree efficiently supports the searches of future positions of moving objects by considering the changes of moving objects included in the node as time-parameter. To maximize space utilization, we propose the new update and split methods. Finally, we perform various experiments to show that our approach outperforms others.

Keyword : Moving Object, Spatio-Temporal Index Structure, Future Position Retrieval

1. 서 론

최근 정보통신사업, 위성사업 그리고 지능형 교통 시스템(ITS : Intelligent Transport System) 사업 등의 위치 기반 서비스에 대한 급속한 발전으로 이동 객체의 위치를 모니터링 할 수 있게 되면서 위성 위치 확인 시스템(GPS : Global Positioning System)과 같은 위치기반 서비스를 제공하는 응용 서비스들이 여러 분야에서 광범위하게 제공되고 있다. 위치기반 서비스의 기술 개발은 광고의 시장성 증가, 홈쇼핑 등 전자상거래의 활성화에 따른 물품 이동 경로의 최적화, 긴급 상황에 대비한 안전 서비스의 수요 증가 및 위치기반 게임과 같은 신세대 취향의 엔터테인먼트 분야에

* 정 회 원 : 충북대학교 정보통신공학과 박사과정
dmseo@netdb.chungbuk.ac.kr(제 1저자)
** 정 회 원 : 충북대학교 정보통신공학과 박사과정
ksbok@netdb.chungbuk.ac.kr(공동저자)
*** 정 회 원 : 충북대학교 전기전자컴퓨터공학부 부교수
yjs@cbucc.chungbuk.ac.kr(공동저자)
**** 정 회 원 : 배재대학교 전자상거래학부 전임강사
bylee@disk.co.kr(공동저자)
☆ 이 논문은 2003년도 한국학술진흥재단의 선도연구자 지원사업(KRF-2003-041-D00489)에 의하여 연구되었음.

파급될 것으로 보이며, 이와 관련하여 지속적으로 움직이는 이동 객체의 위치를 추적하고 관리하기 위한 연구들도 꾸준히 증가할 것으로 예상된다.

일반적으로, 이동 객체 데이터베이스(Moving Object Databases : MOD)에서는 동적인 속성을 가지는 이동 객체에 대한 데이터 유형을 과거, 현재 그리고 미래 위치로 구분하여 표현한다. 대부분 이동 객체에 대한 연구들은 과거의 궤적 또는 현재 위치에 대한 데이터를 관리하는데 중점을 두고 많은 연구가 진행되었다. 최근 미래 위치에 대한 중요성이 증가됨에 따라 이동 객체의 현재 위치를 통해 미래 위치를 예측하기 위한 연구들이 진행 중이다[1].

지속적으로 움직이는 이동 객체의 위치를 추적하고 저장하기 위해서는 새로운 기술의 색인 구조가 요구된다. 기존의 색인 구조는 특정 시점의 객체 위치를 관리하기 때문에 이동 객체의 위치 정보와 같이 동적인 속성을 가지는 정보를 표현, 저장 그리고 질의하는데 있어서 비효율적이다. 그 예로, R-트리[2] 기반의 기존 공간 색인 구조는 지속적으로 변경되는 이동 객체의 위치 정보를 색인하기 위해서 색인 구조에 있어 많은 갱신 비용을 요구하고 성능을 크게 감소시키는 문제점을 가진다.

이러한 문제점을 해결하기 위해 새로운 색인 구조들이 제안되었다. 그 예로, VCI[3]는 다수의 연속적인 범위 질의를 효율적으로 처리하기 위해 기존 R-트리의 각 노드에 V_{max}라는 이동 객체의 최대 속도 값을 유지하고 TPR-트리[4]는 이동 객체의 미래 위치에 대한 질의를 효율적으로 처리하기 위해 시간에 대한 MBR과 속도 벡터를 사용함으로서 이동 객체를 주기적으로 갱신하는데 드는 높은 비용을 줄였다. 그러나 이러한 색인 구조들 또한 객체의 동적 변화에 따라 많은 갱신 비용을 소요할 뿐만 아니라 이로 인해 검색 성능을 저하시키는 문제점이 있다. 또한, LUR-트리[5]에서는 이동 객체의 현재 위치를 색인할 때 발생하는 갱신 비용을 감소시키기 위한 기법을 제안하였다. 그러나 현재 위치에 대해서만 색인을 구성하기 때문에 미래 위치 검색을 제공하지 못한다.

본 논문에서는 이동 객체의 현재 및 미래 위치 검색을 효과적으로 수행하기 위한 TPKDB-트리라는 새로운 색인 구조를 제안한다. 제안하는 색인 구조는 R-트리 기반의 빈번한 갱신 비용을 최소화하기 위해 KDB-트리와 이동 객체의 현재 위치를 직접 연결할 수 있는 보조 인덱스로 구성된다. 제안하는 색인 구조에서 이동 객체의 미래 위치를 효과적으로 검색하기 위해 이동 객체의 현재 위치와 속도 정보를 표현하고 미래 위치를 예측할 때 발생하는 불필요한 영역의 확장을 최소화하기 위해 노드에 존재하는 객체들이 특정 영역을 벗어나는 시간 정보를 함께 표현한다. 또한 공간활용도를 향상시키기 위한 새로운 분할 기법을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 기존에 제안된 이동 객체를 위한 색인 기법들에 대해 살펴보고 기존 색인 구조의 문제점을 해결하기 위한 접근 방향을 서술한다. 3장에서는 제안하는 TPKDB-트리의 구조와 삽입, 분할 그리고 현재와 미래에 대한 검색 기법에 대해 기술한다. 4장에서는 다양한 실험을 통해 기존에 제안된 색인 구조와의 비교 분석을 수행한다. 마지막으로, 5장에서는 결론 및 향후 연구 방향에 대해 기술한다.

2. 관련 연구

기존의 공간 색인 기법에서는 위치가 고정된 객체의 위치를 저장하고 관리하기 위한 많은 연구들이 진행되었으나, 최근에는 이동 객체를 효과적으로 저장하고 관리하기 위한 많은 연구들이 진행되고 있다. 시간에 따라 이동 객체의 위치가 변화하고 누적되는 이동 객체의 궤적 정보는 선형적으로 증가하게 되고, 선형적으로 증가된 대용량의 데이터에 대한 검색은 고비용의 연산을 필요로 한다. 따라서 시간에 따라 증가되는 궤적 정

보와 대용량의 궤적 정보에 대한 효율적 검색을 위해 궤적에 대한 색인이 요구된다. 초기의 시공간 색인 구조는 사분 트리, KDB-트리와 같은 공간 분할 방법과 R-트리 계열의 데이터 분할 방법으로 나뉜다[6].

사분 트리는 이동 객체의 위치를 키 값으로 해쉬하는 방법을 사용하여 비교적 간단한 과정으로 색인이 가능하다는 장점이 있으나 데이터 집합이 비정규 분포일 경우 특정 영역에 계속적인 오버플로우를 발생시켜 색인의 성능이 저하되는 문제가 발생한다. 그리고 이동 객체의 이동성으로 인해 이동 객체 밀집 지역을 빈번히 발생시키고 이로 인해 사분 트리의 성능 저하 문제를 발생시킨다.

최근 LUR-트리[5], Q+R-트리[7], VCI[3] 그리고 TPR-트리[4]와 같은 R-트리 계열의 색인 구조가 제안되었다. LUR-트리는 지속적으로 움직이는 이동 객체의 현재 위치를 색인할 경우 요구되는 많은 갱신 비용을 줄이기 위해 제시된 색인 구조이다. 이러한 색인 구조는 이동 객체의 위치가 갱신되는 동안 불필요한 노드의 분할과 합병을 줄이기 위해서 갱신되는 객체의 정보가 동일 노드 내에서 변경될 경우 현재 노드에만 변경된 정보를 반영한다. 이와 같은 방법을 사용함으로서 색인 구조의 많은 갱신 비용을 줄이고 이동 객체의 현재 위치를 효율적으로 색인 한다. 하지만 이동 객체의 현재 위치 정보만을 색인하지 못하는 문제점이 있다.

Q+R-트리는 이동 객체를 색인하는 기존의 R-트리 기반의 색인 구조가 가지는 많은 갱신 비용을 줄이며 이동 객체의 위치를 효율적으로 색인하기 위해 제안된 색인 구조이다. Q+R-트리는 R*-트리와 사분 트리로 구성된 혼합형 트리로 이동 객체를 이동 패턴과 지리적 특성에 따라 동적 이동 객체와 정적 이동 객체로 구분하여 색인한다. 갱신 비용을 많이 필요로 하는 동적 객체는 다른 색인 구조에 비해 갱신 비용이 우수한 사분 트리에 색인하고 갱신 비용을 많이 필요로 하지 않는 정적 객체는 검색 성능을 향상시키기 위해서 R*-트리에 색인한다. 하지만, 사분 트리의 특성으로 인해 색인되는 객체가 편향될 경우 색인 구조의 크기가 커지고 불균형 트리가 되어 검색 성능이 저하되는 문제점이 있다.

VCI와 TPR-트리는 갱신 비용을 줄임으로서 이동 객체를 효율적으로 색인하는 방법과 이동 객체에 대한 미래 위치 검색을 지원하는 색인 구조이다. VCI는 이동 객체를 색인하는 색인 구조에서 다수의 연속적인 범위 질의를 효율적으로 처리하기 위해 질의 인덱스와 함께 제안된 색인 구조이다. 기존 R-트리를 기반으로 하고 각 노드에 노드 안에 포함되어 있는 객체들 중 가장 큰 속도 값을 취하는 V_{max} 필드를 추가함으로서 이동 객체를 주기적으로 갱신하는데 드는 높은 비용과 많은 갱신 비용으로 검색 성능이 저하되는 기존의 색인 구조의 문제를 해결하고 미래 위치 검색을 제공한다. TPR-트리는 VCI 트리와 유사한 방법으로 R-트리를 기반으로 하며 각 노드의 에지(edge)에 대해 대응되는 방향으로 이동하는 객체들 중 가장 큰 속도 값을 유지한다. 하지만, VCI 그리고 TPR-트리는 R-트리를 기반으로 하기 때문에 이동 객체를 색인하는 공간 색인의 문제점인 갱신 비용 문제와 노드의 MBR을 시간의 함수로 표현함으로서 발생하는 노드들간의 겹침으로 검색 시간이 저하되는 문제점이 있다.

이와 같이, R-트리 계열의 색인 구조는 높이 균형 트리 구조이기 때문에, 데이터의 분포와 관계없이 우수한 성능을 나타낸다. 그러나 이동 객체의 계속되는 위치 이동으로 인해 색인의 변경이 발생하고, 색인의 빈번한 변화로 전체적인 색인의 성능이 저하되는 문제를 발생한다. 위와 같은 문제의 원인은 공간 색인이 정적 데이터 기반으로 설계되고, 검색이나 삽입과 같이 색인의 변경을 위한 연산이 별도로 정의되어 있지 않기 때문이다. 그래서 본 논문에서는 이동 객체를 색인하는 기존 색인 구조들의 많은 갱신 비용을 줄이고 미래 위치 검색 성능을 향상시키기 위해서 이동 객체의 위치 데이터를 3차원 상에서의 점으로

모델링하고 공간 분할 방법을 사용하는 색인 구조인 KDB-트리을 사용하여 색인하는 방법을 제안한다.

그리고 기존의 KDB-트리의 분할 정책은 이동 객체의 특성을 고려하지 않은 분할 정책이고 FD 분할 정책, 데이터 종속적인 분할 정책 그리고 분할 종속적인 분할 정책 모두 노드에 저장된 데이터 분포 편차가 클 경우 불균등한 데이터 분할이 일어나 노드의 공간활용도가 저하되는 문제점을 가진다[8,9,10]. 본 논문에서는 이동 객체를 색인하는 색인 구조에 효율적이면서 기존의 분할 정책이 가지는 중간 노드와 단말 노드의 공간활용도를 개선시키기 위한 새로운 갱신 및 분할 기법을 제안한다.

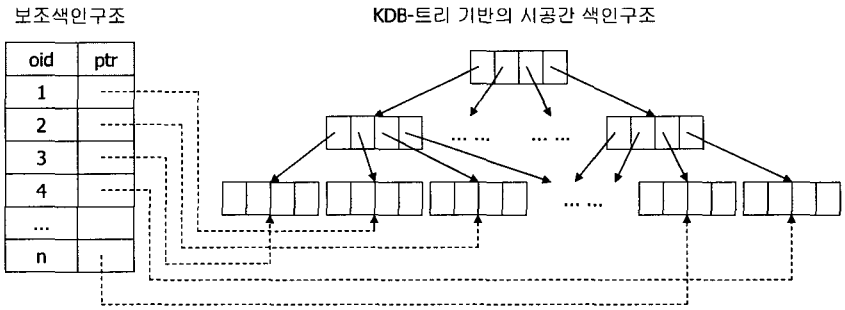
3. TPKDB-트리

제안하는 색인 구조는 갱신 비용을 최소화하여 이동 객체 검색의 효율성을 증가시키고 이동 객체를 선형함수로 표현함으로서 불필요한 갱신을 줄이는 방법을 제안한다. 그리고 노드 내에 포함되어 있는 이동 객체의 변화를 시간에 대한 파라미터로 유지함으로서 효율적으로 이동 객체의 미래 위치 검색을 지원한다. 또한, 공간활용도를 최대화하기 위해 EDD 분할(Enhanced Data Dependent Splitting)과 EFP 분할(Enhanced First Division Splitting) 기법을 제안하는 색인 구조이다. 이 장에서는 제안하는 TPKDB-트리의 구조와 삽입, 삭제, 분할 그리고 검색 알고리즘에 대해 기술한다.

3.1 색인 구조

TPKDB-트리는 불필요한 갱신 비용을 최소화하고 미래 위치에 대한 검색 성능을 향상시키기 위해 그림 1과 같이 KDB-트리와 보조 색인 구조의 혼합형 색인 구조로 구성되어 있다. 제안하는 색인 구조는 R-트리 계열의 색인 구조가 가지는 문제점인 데이터의 삽입과 삭제로 발생하는 빈번한 색인 구조의 변경을 피하고, 노드들의 겹침으로 인한 검색 성능 저하를 개선하고자 KDB-트리를 기반으로 한다. 그리고 보조 색인 구조는 이동 객체의 ID를 별도로 관리하는 색인 구조로 어떤 종류의 색인 구조를 사용해도 무관하나 이동 객체의 빠른 검색을 지원함으로서 갱신 성능을 향상시키기 위해서 이동 객체의 ID 검색 시 디스크 I/O 횟수가 최소인 색인 구조를 사용하고 보조 색인 구조에 색인된 이동 객체 ID는 해당 객체가 색인된 TPKDB-트리의 단말 노드와 연결 정보를 가진다. 본 논문에서는 보조 색인 구조로 해쉬를 사용한다.

TPKDB-트리는 지정된 영역 내에 있는 이동 객체에 대해 색인을 하며, 이 영역은 실세계에서 이동 객체가 이동할 수 있는 지역을 2차원의 값으로 나타낸 것이다. TPKDB-트리에서 이동 객체는



〈그림 1〉 TPKDB-트리의 구조

식 1과 같이 표현된다.

$$<oid,\ t_{ref},\ p_{ref},\ v_{ref}> \quad \text{(식 1)}$$

*oid*는 이동 객체에 대한 식별자, t_{ref}는 현재 시간, p_{ref}는 t_{ref} 시간에서 객체의 위치로 (p_1, p_2, ..., p_n)과 같다. 또한, v_{ref}는 t_{ref} 시간에서 객체의 속도로 (v_1, v_2, ...,v_n)과 같이 나타낸다. 이를 이용하여 미래 시점 t에서 객체의 위치는 식 2와 같이 t_{ref} 시점의 객체 위치와 속도를 통해 계산한다.

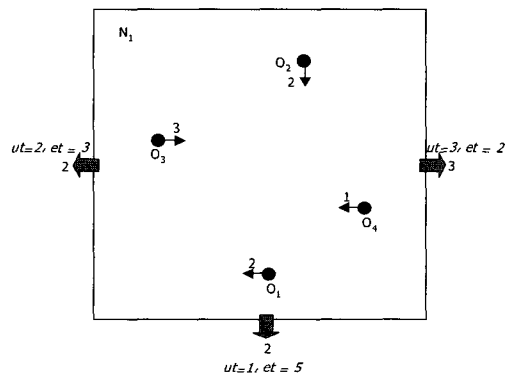
$$p_t = p_{ref} + v_{ref}(t - t_{ref}) \quad \text{(식 2)}$$

TPKDB-트리의 단말 노드는 식 3과 같이 표현된다.

$$<RS,\ t_{upd},\ t_{esc},\ v_{max},\ oid_n,\ t_{refn},\ p_{refn},\ v_{refn}> \quad \text{(식 3)}$$

*RS*는 단말 노드에 존재하는 객체를 포함하는 공간 영역 정보 $(RS_1, RS_2, \ldots, RS_n)$이다. 이때, $RS_i = [p_i^-, p_i^+]$로 p_i^-와 p_i^+는 i번째 차원의 시작 위치와 끝 위치로 단말 노드가 색인하는 위치를 나타낸다. 단말 노드가 색인하는 끝 위치를 나타낸다. t_{upd}는 단말 노드의 갱신 시간으로 단말 노드 내에 존재하는 객체 중에서 가장 오래된 갱신 시간을 기록한다. t_{upd}를 가장 오래된 시간으로 기록하는 것은 미래 위치 검색 과정에서 확장되는 노드 안에 포함되어 있는 모든 객체의 정확한 미래 위치 정보를 획득하기 위해서이다. v_{max}는 단말 노드에 존재하는 객체의 속도값 $(v_{\max_1}, v_{\max_2}, \ldots v_{\max_n})$로 각 축의 양 방향 최대 속도값을 나타낸다. 이때, v_{max}는 $[v_i^-, v_i^+]$로 v_i^-와 v_i^+는 i번째 차원의 왼쪽 및 오른쪽 방향으로 이동하는 객체의 최대 속도를 나타낸다. 또한 t_{esc}는 단말 노드에 존재하는 객체들이 *RS* 영역을 벗어나는 최소 시간으로 미래 위치 검색 과정에서 불필요한 영역 확장을 없애기 위해 사용한다. KDB-트리 기반의 색인 구조는 영역을 분할하여 색인을 구성하기 때문에 사장 공간(dead space)이 존재한다. 한 사장 공간 내에서 이동하는 객체에 대한 미래 위치를 검색하는데 불필요한 영역 확장을 없애기 위해 t_{esc}을 사용한다. 만약 미래 위치 검색이 t_{upd}를 기준으로 t_{esc} 시간 내에서 수행된다면 TPR-트리나 VCI -트리처럼 영역을 확장하는 것이 아니라 현재 영역을 미래 위치까지 확장된 영역으로 판별하여 검색을 수행한다. 이동 객체의 실제 위치를 나타내는 단말 노드 엔트리는 $<oid,\ t_{ref},\ p_{ref},\ v_{ref}>$이다. oid, t_{ref}, p_{ref}, v_{ref}는 이동 객체의 정보와 동일하다.

그림 2에서 각 점은 단말 노드에 색인된 이동 객체를 나타낸다. 점과 함께 표현되는 화살표는 이동 객체가 움직이는 방향을 나타내며, 화살표 옆에 표현되는 숫자는 이동 객체의 속도를 나타낸다. 예를 들어, 이동 객체 O_2는 속도 2를 가지고 남쪽으로 이동하는 것을 나타낸다. 단말 노드 각 에지에 표현된 화살표는 대응되는 방향으로 이동 중인 객체가 한 개 이상 있다는 것을 나타낸다. 화살표와 함께 표현되는 숫자는 화살표와 동일한 방향으로 이동하는 객체들 중에 가장 빠르게 움직이는 이동 객체의 속도를 의미한다. 그리고 화살표와 함께 표현되는 *ut*와 *et*는 화살표와



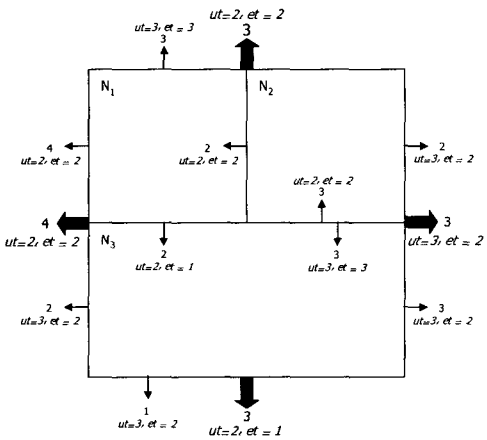
〈그림 2〉 TPKDB-트리의 단말 노드

동일한 방향으로 움직이는 객체에 대해 단말 노드가 가지는 t_{upd}와 t_{esc}를 의미한다. 즉, 그림 2의 단말 노드에 대해 왼쪽으로 향하는 화살표와 '3'은 노드 확장 시 왼쪽으로 단위 시간에 대해 3만큼 영역을 확장한다는 것을 나타내고 $up(t_{upd})$=3와 $et(t_{esc})$=2는 미래 질의 처리를 위해 노드 확장 시 질의 시간 5까지는 노드 확장이 불필요하다는 것을 나타낸다.

TPKDB-트리의 중간 노드 구조는 식 4와 같이 표현된다.

$$<RS,\ t_{upd},\ t_{esc},\ v_{max},\ RS_n,\ t_{updn},\ v_{maxn},\ t_{escn},\ ptr_n> \quad \text{(식 4)}$$

RS는 중간 노드에 존재하는 단말 노드를 포함하는 공간 영역 정보를 나타내고 t_{upd}는 중간 노드의 갱신 시간으로 중간 노드 내에 존재하는 단말 노드의 헤더 정보 중에서 가장 과거의 갱신 시간을 기록한다. 그리고 $v_{\max}$는 중간 노드에 존재하는 단말 노드의 최대 속도값을 기록하고 t_{esc}는 중간 노드에 존재하는 단말 노드들이 RS 영역을 벗어나는 최소 시간을 기록한다. 단지, 단말 노드는 객체의 실제 값을 이용하여 헤더를 구성하지만 중간 노드는 하위 노드들에 존재하는 헤더 정보를 이용하여 헤더 정보를 구성한다. 그리고 중간 노드에 존재하는 엔트리는 하위 노드에 존재하는 자식 노드의 헤더를 이용하여 구성된다. 또한, 엔트리의 ptr_i은 단말 노드 N_i가 저장되어 있는 곳의 포인터를 나타낸다.



〈그림 3〉 TPKDB-트리의 중간 노드

그림 3은 제안하는 색인 구조의 중간 노드 구조를 나타낸 것이다. 그림 3의 중간 노드는 단말 노드 N_1, N_2 그리고 N_3을 포함한다. 단말 노드 N_1, N_2 그리고 N_3은 각각 해당 노드에 포함되어 있는 이동 객체들에 의해 단말 노드의 헤더 정보인 $〈RS,\ t_{upd},\ v_{\max},\ t_{esc}〉$가 구성된다. 그리고 중간 노드의 헤더 정보인 $〈RS,\ t_{upd},\ v_{\max},\ t_{esc}〉$는 중간 노드에 포함되어 있는 단말 노드의 $〈RS,\ t_{upd},\ v_{\max},\ t_{esc}〉$에 의해 구성된다. 예를 들어, 단말 노드 N_2와 N_3가 각각 동쪽으로 향하는 객체에 대해 2와 3의 $v_{\max}$을 가진다. 그러므로 중간 노드의 헤더 정보 중에서 동쪽으로 향하는 객체에 대한 $v_{\max}$는 가장 빠른 속도 3을 취한다. 그리고 t_{upd}, t_{esc}는 가장 작은 값을 취하기 때문에 $up(t_{upd})$=3과 $et(t_{esc})$=2를 취하게 된다. 즉, 중간 노드에서 동쪽으로 향하는 객체에 대한 중간 노드의 $up(t_{upd})$=3와 $et(t_{esc})$=2는 미래 질의 처리를 위해 중간 노드 확장시 질의 시간 5까지는 동쪽으로 노드 확장이 불필요하다는 것을 나타낸다.

3.2 삽입 알고리즘

TPKDB-트리에서의 삽입·삭제 알고리즘은 KDB-트리의 삽입·삭제 알고리즘과 유사하다. 그러나 이동 객체의 갱신 처리를 위해 수반되는 삽입·삭제 연산 시 보조 색인 구조를 통한 빠른 객체 검색과 갱신 전과 갱신 후의 이동 객체의 위치가 동일 노드이고 이동 객체의 궤적을 구성하는 파라미터들의 변경이 없는 경우 갱신을 피하는 방법을 사용함으로서 갱신 성능을 향상시키는 방법들이 사용되므로 이와 관련된 연산이 기존 알고리즘에 추가되었다.

이동 객체에 대한 삽입은 해쉬와 TPKDB-트리, 두 색인 구조의 삽입 연산을 수반한다. 그림 4는 TPKDB-트리의 삽입 알고리즘을 나타낸다. 먼저, SearchBucket( ) 함수를 호출해서 삽입 연산을 요청한 이동 객체의 *oid*를 해쉬하는 버켓을 찾는다. 버켓이 선택되면, FindLeafNode( )를 통해 *oid*와 연결된 TPKDB-트리의 단말노드가 있는지를 검사한다. TPKDB-트리의 단말노드가 획득되면 삽입

```
Algorithm Insert()
입력값
 mObj : 삽입 객체의 정보를 유지하는 구조체
 1 : {
 2 : SearchBucket(mObj, ptrBucket); /* 해쉬 버켓의 포인터 획득 */
 3 : if(leaf = FindLeafNode(mObj, ptrBucket)) { /* 단말 노드 포인터 획득 */
 4 : if(!(ChgLeafPos(mObj, leaf))) { /* 객체의 변경 위치 검사 */
 5 :  Update(mObj, leaf); /* 기존 노드 안에서의 위치 변경 */
 6 :  if(!(ChgLeaf(mObj, leaf))) { /* 단말 노드의 변경사항 검사 */
 7 :  /* 객체가 삽입될 단말 노드의 전체 경로 검색 */
 8 :  SearchNodePath(mObj, leaf, pathQueue);
 9 :  Update(leaf, pathQueue); } } /* 중간 노드의 정보 변경 */
10 :  else {
11 :  Delete(mObj, leaf); /* 다른 노드로의 위치 변경 */
12 :  SearchNodePath(mObj, leaf, pathQueue);
13 :  if(leaf.EntryNum < MAXENTRY) /* 오버플로우 검사 */
14 :  Insert(mObj, leaf, pathQueue); /* 단말 노드에 삽입 연산 */
15 :  else
16 :  Split(mObj, leaf, pathQueue); /* 분할 연산 */
17 :  /* 보조 인덱스와 TPKDB-트리의 단말노드 연결 */
18 :  LinkLeaf(ptrBucket, leaf); } /* 버켓과 단말 노드 연결 */
19 : }
20 : else {
21 : if(ptrBucket ->entryNum < HASH_MAXENTRY)
22 :  Insert(mObj, ptrBucket); /* 버켓에 OID 삽입 */
23 : else
24 :  SplitBucket(mObj, ptrBucket); /* 오버플로우 발생한 버켓 분할 */
25 : SearchNodePath(mObj, leaf, pathQueue);
26 : if(leaf.EntryNum < MAXENTRY)
27 :  Insert(mObj, leaf, pathQueue);
28 : else
29 :  Split(mObj, leaf, pathQueue); /* 오버플로우 발생시 분할 처리 */
30 : LinkLeaf(ptrBucket, Leaf);
31 : }
32 : }
```

〈그림 4〉 TPKDB-트리의 삽입 알고리즘

을 요청한 이동 객체는 기존 정보의 갱신을 요청하는 것이므로 그림 4의 4행 부분을 실행한다. 반면, TPKDB-트리의 단말노드가 획득되지 않으면 색인 구조에 이동 객체의 정보를 처음 삽입하는 것이므로 그림 4의 20행 부분을 실행한다.

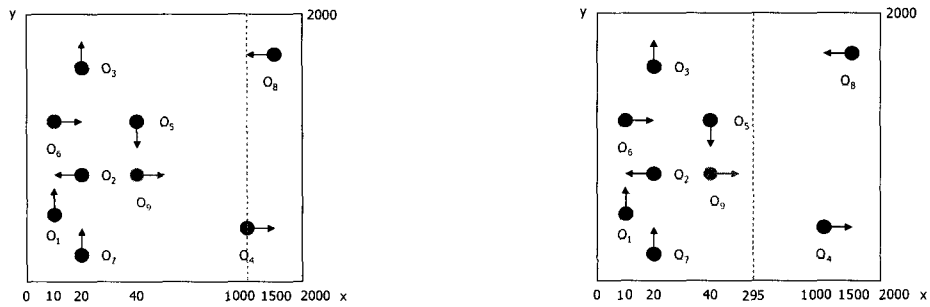
기존 정보의 갱신 연산은 갱신된 이동 객체의 위치가 TPKDB-트리의 이전 단말 노드에서 다른 단말 노드로의 변경이면 갱신 전의 이동 객체를 삭제 후 갱신된 이동 객체를 새로 삽입하는 방법을 취한다. 또한, 삽입하려는 *oid*가 선택된 버켓에 없다면 삽입 연산을 요구한 이동 객체의 *oid*를 해쉬의 선택된 버켓에 삽입한 후 TPKDB-트리의 삽입 연산을 통해서 이동 객체가 요청한 삽입 연산을 수행한다. TPKDB-트리의 삽입 연산은 기존의 KDB-트리의 삽입 연산과 유사하다. 먼저, TPKDB-트리의 검색 연산을 통해서 삽입 연산을 요청한 이동 객체가 삽입될 단말 노드를 찾는다. 이동 객체가 삽입될 단말 노드를 선택한 후 새로 삽입될 이동 객체에 의해 선택된 단말 노드가 분할이 발생하는지 검사한다. 만약 분할이 발생하지 않는다면 선택된 단말 노드에 삽입하려는 이동 객체를 삽입하고 삽입된 이동 객체로 인한 단말 노드의 t_{upd}, t_{esc} 그리고 v_{max}가 변경되는지 검사한다. 이동 객체 삽입 후 단말 노드의 t_{upd}, t_{esc} 그리고 v_{max}의 변경이 발생하면 변경된 값을 단말 노드에 반영해 주고 이 변경으로 인한 해당 단말 노드의 부모 노드 변경 사항을 검사 후 반영한다. 이와 같은 노드 정보의 변경은 루트 노드까지 반영될 수 있다. 단말 노드에 이동 객체를 삽입 후 분할이 발생한다면 TPKDB-트리의 분할 연산을 통해서 분할이 발생한 단말 노드를 처리해 주고, 분할로 인한 t_{upd}, t_{esc} 그리고 v_{max}의 변경을 요청하는 단말 노드와 중간 노드의 변경을 반영한다. 단말 노드와 중간 노드의 분할 방법에 대해서는 3.3절 분할 알고리즘에서 기술하겠다.

해쉬와 TPKDB-트리의 단말 노드를 포인터로 연결함으로서 *oid*를 통한 이동 객체의 빠른 검색과 이동 객체의 갱신 비용을 효과적으로 줄이기 위해서, 이동 객체의 삽입 요청에 의해 해쉬에 이동 객체의 *oid*를 삽입하고 TPKDB-트리에 이동 객체를 삽입한 후에는 이동 객체가 삽입된 TPKDB-트리의 단말 노드와 이동 객체의 *oid*가 삽입된 해쉬의 버켓을 포인터로 연결한다.

삭제 연산은 이동 객체의 갱신, 분할 그리고 합병에 따른 재 삽입을 위해 기존 이동 객체를 삭제하는 경우에 요청된다. 삭제하려는 이동 객체를 TPKDB-트리의 단말 노드에서 삭제 후, 삭제된 이동 객체를 포함하고 있던 단말 노드에서 언더플로우(Underflow)가 발생하는지를 검사한다. 언더플로우가 발생하지 않는다면 삭제된 이동 객체에 의해 해당 단말 노드의 t_{upd}, t_{esc} 그리고 v_{max}가 변경되는지 검사한다. 이동 객체 삭제로 인한 단말 노드의 t_{upd}, t_{esc} 그리고 v_{max}가 변경이 발생하지 않는다면 삭제 연산을 마치고, 변경이 발생하면 변경 사항을 루트 노드까지 반영한다. 만약 삭제된 이동 객체에 의해 해당 단말 노드가 언더플로우가 발생한다면 합병 연산을 통해 언더플로우가 발생한 단말 노드를 주변 노드와 합병함으로서 색인 구조의 공간활용도를 높인다.

3.3 분할 알고리즘

시공간 색인으로 KDB-트리을 이용할 경우 KDB-트리가 공간분할 방식이기 때문에 계속적인 데이터의 삽입으로 인해 단말 노드에 오버플로우가 발생하고 이로 인해 단말 노드가 분할된다. 단말 노드 분할 정책에는 데이터 종속적인 분할과 분할 종속적인 분할이 있다. 고정된 공간을 색인하는 KDB-트리가 광범위한 공간을 색인하면서 일부 영역으로 객체가 편향되어 색인되는 경우 두 분할 정책 모두 공간활용도 저하 문제를 가질 수 있다. 그림 5는 오버플로우가 발생한 단말 노드의 분할을 보여준다. 단말 노드가 수용할 수 있는 최대 엔트리 수를 8이라고 가정했을 때 이동 객체 O_9의 삽입으로 인해 단말 노드에 오버플로우가 발생하는 것을 보여준다.

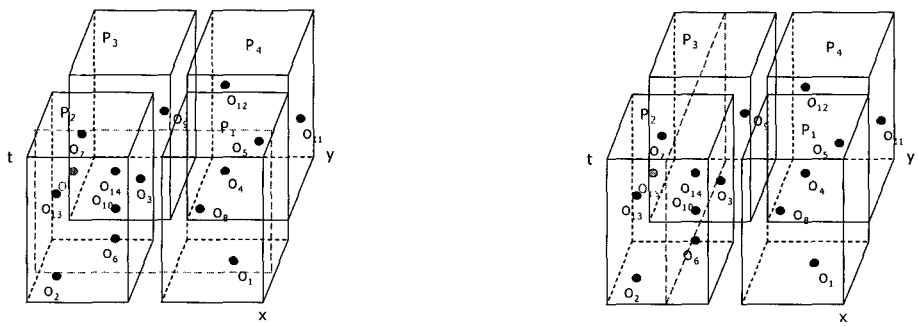


(a) 분할 종속적인 분할 예 (b) 데이터 종속적인 분할 예

〈그림 5〉 오버플로우 발생한 단말노드 분할

그림 5의 (a)에서는 분할 종속적인 분할로 인한 단말 노드 분할 예를 보여주고 있고 편향된 객체를 포함하고 있는 단말 노드의 분할 종속적인 분할로 인해 공간활용도 저하 문제가 발생하는 것을 확인할 수 있다. 그림 5의 (b)에서는 데이터 종속적인 분할로 인해 단말 노드 분할 예를 보여주고 있고 데이터 종속적인 분할 또한 그림 5의 (a)에서와 마찬가지로 편향된 객체를 포함하는 경우 분할로 인해 공간활용도 저하 문제가 발생하는 것을 확인할 수 있다.

계속적인 단말 노드 분할로 인해 중간 노드가 오버플로우 되면 중간 노드를 분할한다. 중간 노드 분할 정책에는 FS(Forced Splitting) 분할과 FD (First Division) 분할이 있다. FS 분할은 강제적 분할 전파 문제로 인해 단말 노드를 과도하게 분할한다. 중간 노드가 강제적으로 하위의 단말 노드에 분할을 전파하게 되면 단말 노드들의 공간 활용도가 저하되는 문제를 가진다. 이 문제는 이동 객체의 특성을 색인하는 KDB-트리의 단말 노드에서 더욱 심각한 공간활용도 저하 문제를 발생시킨다. 그림 6은 FS 분할로 인한 단말 노드들의 강제 분할 예를 보여주고 있다. 단말 노드가 수용할 수 있는 최대 단말 노드의 수를 5라고 가정했을 때, 이동 객체 O_2, O_3, O_6, O_{13} 그리고 O_{14}를 수용하는 단말 노드 P_2는 이동 객체 O_{15}의 삽입으로 오버플로우가 발생한다. 오버플로우가 발생한 단말 노드 P_2는 분할을 통해 오버플로우 문제를 해결한다. 그림 6에서 (a)는 단말 노드 P_2의 오버플로우로 인해 공간 도메인 x축을 기준으로 강제 분할된 예를 보여주고 (b)는 공간 도메인 y축을 기준으로 강제 분할된 예를 보여주고 있다. FS 분할 정책에 의해 단말 노드를 분할할 경우



(a) 공간 도메인 x로 강제 분할된 예 (b) 공간 도메인 y로 강제 분할된 예

〈그림 6〉 단말노드의 FS 분할

오버플로우가 발생한 단말 노드 이외에도 강제적으로 분할이 발생하는 것을 확인할 수 있다. 그리고 (a)에서 강제 분할이 발생한 단말 노드 P_3와 (b)에서 강제 분할이 발생한 단말 노드 P_1과 같이 분할 시점이 아닌 다수의 단말 노드가 분할되어 색인의 크기를 증가시키고 공간활용도 저하 문제가 발생하는 것을 확인할 수 있다.

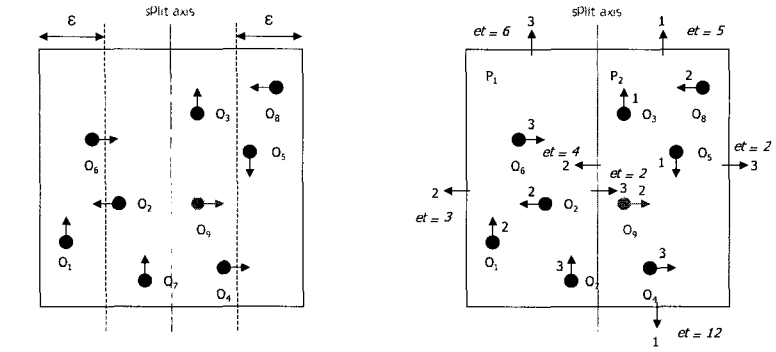
FD 분할은 오버플로우가 발생한 중간 노드의 첫 분할 축을 기준으로 중간 노드를 분할하는 방법이다. 중간 노드를 분할할 때 하위 영역에 대해 강제적으로 분할 전파가 일어나는 것을 제거함으로서 불필요한 단말 노드의 분할을 감소시켜서 다차원 색인에 있어 FS 분할을 사용하는 것보다 성능이 좋은 것으로 연구되었다. 하지만, FD 분할에서도 엔트리가 특정 영역에 편향하는 경우 처음 분할이 발생한 분할 축을 기준으로 일부 영역에 단말 노드와 중간 노드가 편향하므로 색인의 크기가 증가하고 공간활용도 저하 문제가 발생하는 것을 확인할 수 있다.

TPKDB-트리의 분할은 데이터 종속적인 분할과 FD 분할을 기반으로 하고 기존 색인 분할보다 공간활용도가 개선된 분할 정책을 제시한다. 본 논문에서는 단말 노드 분할 정책으로 EDD 분할 (Enhanced Data Dependent splitting) 정책과 중간 노드 분할 정책으로 EFD 분할 (Enhanced First Division Splitting) 정책을 제안한다. TPKDB-트리는 공간분할 방식이기 때문에 계속적인 이동 객체의 삽입으로 인해 단말 노드에 오버플로우가 발생하면 단말 노드에 분할이 일어난다. 기존의 데이터 종속적인 분할의 공간활용도 저하 문제를 해결하기 위해 EDD 분할 정책에서는 개선된 데이터 종속적인 분할을 사용한다. 기존의 데이터 종속적인 분할은 오버플로우가 발생한 단말 노드의 분할 도메인을 순환적 방법에 의해 선정하고, 선정된 분할 도메인에 대응되는 모든 데이터들의 도메인 평균값을 분할 위치로 선정한다.

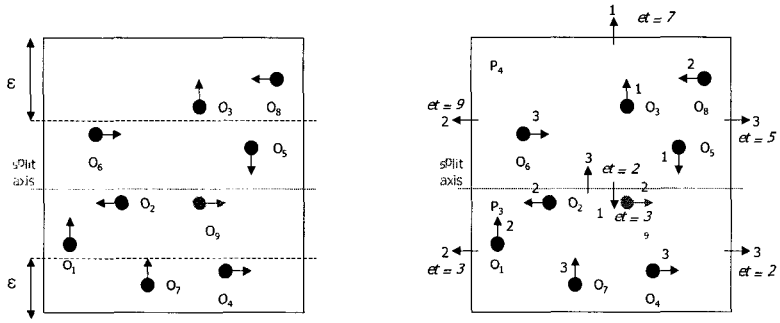
EDD 분할 정책에서는 단말 노드 오버플로우 발생시 데이터 종속적인 분할을 사용하여 모든 도메인에 대한 분할 위치를 선정한다. 분할 위치를 선정할 때는 분할되는 두 단말 노드가 수용해야 하는 최소 엔트리 수에 참여되는 데이터를 제외한 데이터들의 도메인 평균값을 분할 위치로 선정한다. 분할에 참여하는 단말 노드들의 최소 엔트리 수를 보장함으로서 편향된 데이터를 포함하는 오버플로우 단말 노드 공간활용도 저하 문제를 해결할 수 있다. 각 도메인에 대한 분할 위치를 선정 후, 각 단말 노드의 t_{upd}, t_{esc} 그리고 v_{max}를 구한다. 그리고 임의의 t (>각 단말 노드의 t_{esc} 중 가장 큰 값)에 대해 각 단말 노드를 확장한 후, 상대적으로 작게 확장되는 단말 노드를 포함하는 분할 도메인을 분할 도메인으로 선정한다.

그림 7은 EDD 분할로 인한 단말 노드 분할을 보여주고 있다. 단말 노드가 수용할 수 있는 엔트리 수가 8이고 최소 엔트리 수를 2라고 가정한다. 이동 객체 O_9의 삽입으로 단말 노드가 오버플로우 되며 단말 노드가 분할된다. 먼저, EDD 분할은 분할 위치를 선정하고 분할된 각 단말 노드의 t_{upd}, t_{esc} 그리고 v_{max}를 구한다. 그림 7의 (a)에서는 공간 도메인 x축에 대한 단말 노드 분할 예를 보여주고 있다. 그림 7의 (b)에서는 공간 도메인 y축에 대한 단말 노드 분할 예를 보여주고 있다. 공간 도메인 x 축에 대한 분할 위치를 선정할 때 분할될 각 노드의 최소 엔트리 수를 보장하기 위해 이동 객체 O_2, O_3, O_4, O_7 그리고 O_9의 도메인 x축의 평균값을 분할 위치로 선정하고, 도메인 y축에 대한 분할 위치를 선정할 때는 이동 객체 O_1, O_2, O_5, O_6 그리고 O_9의 도메인 y축의 평균값을 분할 위치로 선정한다.

오버플로우 단말 노드를 공간 도메인 x축으로 분할할 경우 단말 노드 P_1과 P_2로 분할되고 공간 도메인 y축으로 분할할 경우 단말 노드 P_3와 P_4로 분할된다. 분할된 각 단말 노드 P_1, P_2, P_3와 P_4 중에 단말 노드 P_2가 가장 큰 t_{upd2}=12을 유지하므로, 단말 노드 P_1, P_2, P_3와 P_4를 임의의 t (>(t_{upd2}=12))에 대해 확장을 한다. 확장된 각 단말 노드 P_1, P_2, P_3와 P_4의 확장 비율을 비교하면 단



(a) x축에 의한 단말 노드 분할



(b) y축에 의한 단말 노드 분할

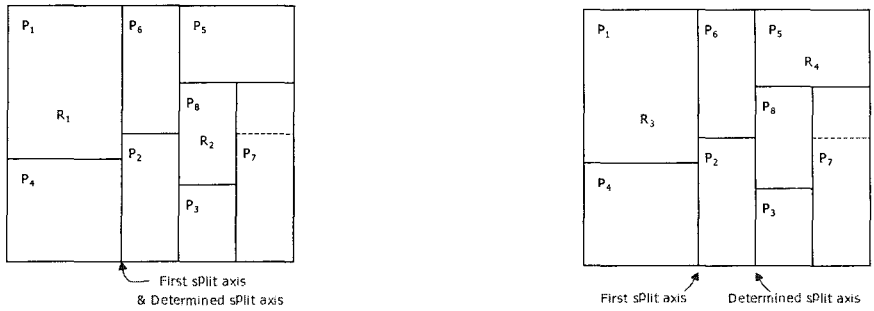
〈그림 7〉 단말 노드의 EDD 분할

말 노드 P_1과 P_2의 확장 비율이 단말 노드 P_3와 P_4에 비해 작으므로 공간 도메인 x축에 대한 분할을 선택한다.

계속적인 단말 노드의 분할로 인해 중간 노드에 오버플로우가 발생하면 중간 노드에 분할이 일어난다. EFD 분할은 중간 노드 분할시 단말 노드의 강제 분할을 막기 위해 FD 분할을 사용하고 공간활용도를 개선하기 위해서 분할 축을 오버플로우 중간 노드의 첫 분할 도메인으로 선정하는 것이 아니라 첫 분할 도메인을 기준으로 분할했을 때보다 공간활용도를 개선시키는 분할 도메인을 분할 축으로 선정한다. 첫 분할 도메인에 비해 공간활용도를 개선시키는 분할 도메인이 없다면 FD 분할과 마찬가지로 첫 분할 도메인을 분할 축으로 선정한다.

그림 8은 FD 분할과 EFD 분할로 인한 중간 노드의 분할을 보여준다. 중간 노드의 최대 수용 엔트리 수를 8이라고 가정한다. 오버플로우 단말 노드의 분할로 인한 오버플로우 중간 노드가 발생한다. 그림 8의 (a)에서는 FD 분할로 인한 오버플로우 중간 노드의 분할을 보여주고 있다. FD 분할은 오버플로우 중간 노드의 첫 분할 도메인을 도메인 축으로 선정하고 분할하기 때문에 단말 노드 P_1과 P_4를 포함하는 중간 노드 R_1과 단말 노드 P_2, P_3, P_5, P_6, P_7 그리고 P_8을 포함하는 R_2로 분할된다.

첫 분할 도메인을 기준으로 일부 영역에 단말 노드가 편향된 오버플로우 중간 노드를 FD 분할을 사용하여 분할할 경우 분할된 중간 노드에 균등하게 단말 노드가 분할되지 않아 공간활용도가 저하되는 것을 볼 수 있다. 그림 8의 (b)에서는 EFD 분할로 인한 오버플로우 중간 노드의 분할



(a) FD 분할 (b) EFD 분할

〈그림 8〉 중간 노드의 EFD 분할

을 보여주고 있다. EFD 분할은 오버플로우 중간 노드의 첫 분할 도메인을 분할 축으로 선정하는 것이 아니라 오버플로우 중간 노드에 포함되어 있는 단말 노드들을 균등하게 분할하는 분할 도메인을 분할 축으로 선정한다. 그림 8의 (b)에서와 같이 오버플로우 중간 노드를 단말 노드 P_1, P_2, P_4 그리고 P_6을 포함하는 중간 노드 R_3와 단말 노드 P_3, P_5, P_7 그리고 P_8을 포함하는 중간 노드 R_4로 균등 분할하는 분할 도메인을 분할 축으로 선정한다.

3.4 검색 알고리즘

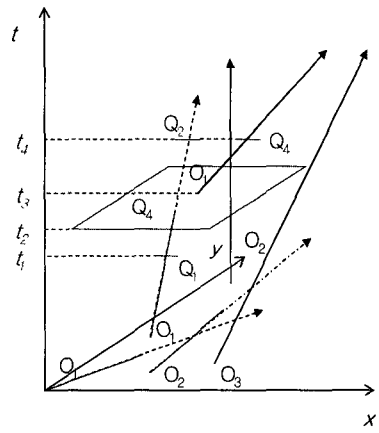
일반적으로, 이동 객체 색인 구조에서는 동적인 속성을 가지는 이동 객체에 대한 데이터 타입을 과거 데이터, 현재 데이터 그리고 미래 데이터로 구분하고 각 데이터 타입에 대한 검색을 제공한다. 지금까지는 이동 객체의 과거와 현재 위치에 대한 데이터를 관리하는데 중점을 두고 많은 연구가 진행되었으며 최근에 와서야 이동 객체의 미래 위치에 대한 데이터 관리의 중요성이 증가하였고 그와 관련된 연구가 활발히 진행 중이다. 본 논문에서 제안하는 TPKDB-트리는 이동 객체의 현재 위치에 대한 검색 처리뿐만 아니라 향후 위치에 대한 검색도 처리하는 색인 구조이다.

TPKDB-트리에서 제공되는 검색의 종류는 두 가지로 객체 질의와 범위 질의가 있다. 각 질의는 식 7와 같이 표현된다.

- 객체 질의 : Q = (*oid*, *t*) (식 7)
- 범위 질의 : Q = (*R*, *t*)

여기서, *oid*는 검색하고자 하는 이동 객체의 아이디를 나타내고 *t*는 검색하고자 하는 시간에 대한 값으로 TPKDB-트리가 이동 객체의 현재와 향후 위치에 대한 검색만을 제공함으로 항상 현재 시간보다 이후의 시간 값을 가진다. R은 검색하고자 하는 영역에 대한 값으로 노드의 영역 정보와 마찬가지로 2차원 값으로 표현된다.

객체 질의는 특정 이동 객체에 대한 현재와 향후 위치에 대한 질의이다. 그 예로는, "28번 시내버스의 현재 위치를 검색해라" 또는 "24번 시내버스의 두 시간 뒤의 위치를 검색해라"와 같은 질의가 있다. 범위 질의는 현재 또는 향후 시간에 대해 정의된 영역 안에 있는 모든 이동 객체에 대한 질의이다. 그 예로는, "현재 운동장 내에 있는 이동 객체를 검색해라" 또는 "앞으로 세 시간 뒤에 정문 앞에 있는 이동 객체를 검색해라"와 같은 질의가 있다. 그림 9는 TPKDB-트리에서 제공하는 여러 종류의 질의의 예를 나타낸 것이다. TPKDB-트리는 색인된 이동 객체들의 정보를 가지고 간단한 선형 함수를 만들어 이동 객체의 궤적을 만들 수 있다. 그림 9에 있는 화살표는 이동 객체 O_1, O_2 그리고 O_3에 대한 정보를 가지고 이



〈그림 9〉 TPKDB-트리의 질의 종류

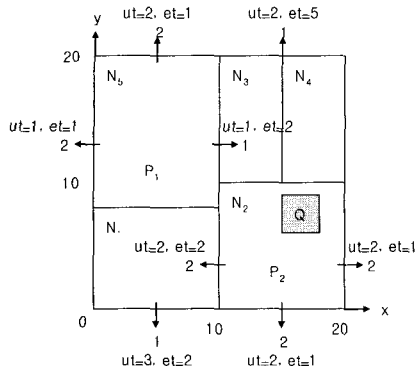
동 객체들의 궤적을 나타낸 것이고 이동 객체 O_1과 O_2 같이 지속적으로 이동 객체의 정보가 변경될 수 있기 때문에 시간에 따라 이동 객체의 궤적은 변경될 수 있다. Q_1, Q_2 그리고 Q_3는 이동 객체 O_1에 대한 객체 질의 그리고 Q_4는 범위 질의를 나타낸 것이다. 색인 구조의 현재 시간 t_{cur}이 t_1이고 이동 객체 O_1에 대한 객체 질의를 요청하면 Q_1과 겹치는 O_1의 궤적에 대한 값을 결과로 얻을 수 있다. 또한, t_{cur}이 t_2이고 범위 질의인 Q_2를 요청하면 Q_2의 질의 영역을 나타내는 사각형 안에 포함된 모든 궤적에 대한 이동 객체를 결과로 얻을 수 있다. Q_2에 대한 질의의 결과로 이동 객체 O_1과 O_2를 결과로 얻는다. 이동 객체의 궤적이 이동 객체를 나타내는 t_{ref}, v_x, v_y, x_{ref} 그리고 y_{ref}에 종속적이기 때문에 질의 요청 시 질의를 내린 시간에 따라 상이한 결과 값을 얻을 수 있다. 일례로, t_{cur}이 t_3보다 작은 시점에서, t_{cur}보다 향후 시간인 t_4에 대한 이동 객체 O_1에 대해 객체 질의 시 Q_2에 대한 결과를 얻고 t_{cur}이 t_3보다 큰 시점에서 동일한 질의에 대해서는 Q_3에 대한 결과를 얻는다.

일반적으로, 객체 질의는 보조 색인 구조를 사용하여 빠르게 검색할 수 있고 현재 시간에 대한 범위 질의는 기존 KDB-트리에서 일정 질의 영역 안에 있는 객체를 검색하는 방법과 동일하다. 단지, TPKDB-트리에서는 향후 시간에 대한 범위 질의를 제공하기 때문에 기존 KDB-트리의 각 노드에 t_{upd}와 t_{esc} 같은 시간 정보와 속도 정보 v_{max}가 포함된다. 그림 10에서는 TPKDB-트리에서 이동 객체의 향후 위치에 대한 검색 방법을 보여주고 있다. 그림 10의 (a)에서는 t_{cur}=3 일 때, TPKDB-트리의 중간 노드들의 구조와 t_{query}=5에 대한 범위 질의가 요청된 것을 나타낸다. TPKDB-트리는 요청된 질의를 처리하기 위해서 중간 노드 P_1과 P_2의 t_{upd}, t_{esc} 그리고 v_{max}을 가지고 t_{query}=5에 대해 노드를 확장한다. 각각의 노드의 확장은 식 8에 의해 구할 수 있다.

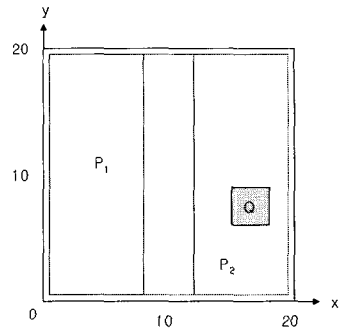
$$ES = \begin{cases} RS & , \text{ if } t_{query} \le (t_{upt} + t_{esc}) \\ RS + (t_{query} - (t_{upd} + t_{esc})) \times v_{\max} & , \ else \end{cases}$$

(식 8)

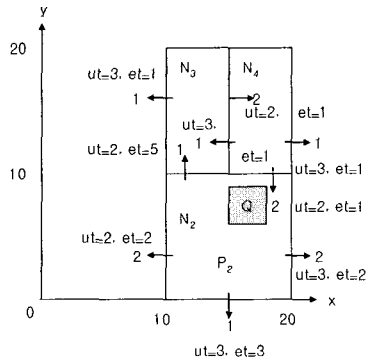
또한, TPKDB-트리가 색인 하는 전체 영역 이상에 대한 확장은 무의미하므로 노드 확장 시 TPKDB-트리가 색인 하는 전체 영역까지만 확장을 한다. 그림 10의 (b)에서는 t_{query}=5에 대해 중간 노드를 확장한 것을 보여준다. 중간 노드 P_1과 P_2를 확장하고 범위 질의의 질의 영역과 비교하였을 때, 중간 노드 P_2만이 질의 영역과 겹침이 발생하기 때문에 중간 노드 P_2에 있는 단말 노드들만을 t_{query}=5에 대해 확장을 한다. 그림 10의 (c)는 미래 시간에 대한 질의 처리를 의해 중간 노드들을 확장 후 미래 질의 영역과 겹침이 발생하는 중간 노드 P_1만을 나타낸 것이다. 미래 질의 영역 안에 있는 이동 객체들을 찾기 위해 중간 노드 P_1에 포함된 단말 노드들을 확장한다. 그림 10의 (d)는 t_{query}=5에 대해 확장된 단말 노드들을 보여주고 있으며, 미래 질의 영역과 겹침이 발생하는 단말 노드는 N_2와 N_4인 것을 알 수 있다. 마지막으로, N_2와 N_4에 있는 이동 객체들의 궤적을 이용하여 질의 영역과 겹침이 발생하는 이동 객체들을 반환함으로서 요구된 질의를 처리할 수 있다.



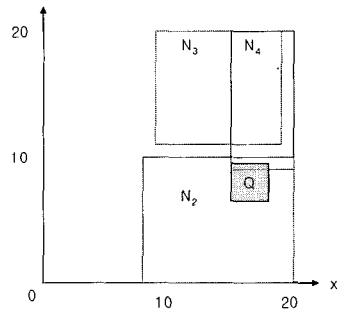
(a) tcur=3 일 때, 중간노드 구조



(b) tquery=5 일 때, 중간노드 확장



(c) tcur=3 일 때, 단말노드 구조



(d) tquery=5 일 때, 단말노드 확장

〈그림 10〉 TPKDB-트리의 미래 위치 검색

4. 실험 및 성능 평가

이 장에서는 제안한 TPKDB-트리를 가지고 여러 실험을 통해 성능 평가한 결과에 대해 기술한다. 먼저, 성능 평가를 위해 사용된 실험 환경과 파라미터들에 대해 기술하고 성능 평가한 결과에 대해 기술한다.

4.1 실험 환경

성능평가에 사용된 시스템은 펜티엄-IV 1.7㎓ 프로세서에 256Mbyte의 메모리를 가지며, 운영체제는 윈도우 2000을 사용하였다. 성능평가에 사용된 데이터 집합은 GiST[11]를 기반으로 한 균등 분포된 100,000개 데이터 개수를 변경시키며 수행하였고, 또한 색인 구성 시 한 노드의 크기는 4Kbytes로 설정하였다. 사용되는 데이터는 이동 객체의 특성인 위치 정보와 속도 정보를 가져야 하기 때문에 1000×1000㎞의 2차원 영역을 랜덤 함수를 이용해 추출된 속도와 방향을 가지고 움직이는 이동 객체로 가정하였다. 이동 객체가 가지는 속도는 1, 1.5, 2, 2.5 그리고 3Km/min (60, 90, 120, 150 그리고 180km/h)로 가정하였다.

본 논문에서 제안하는 TPKDB-트리는 이동 객체의 계속되는 위치 이동으로 색인의 변경이 발생하고, 색인의 빈번한 변경으로 전체적인 색인의 성능이 저하되는 기존 R-트리 계열의 색인 구조 문제를 해결하기 위해 제안된 색인 구조이다. 그래서 제안한 색인 구조의 성능을 평가하기 위해서 본 실험에서는 R-트리 계열의 데이터 분할 방

식을 사용하여 이동 객체를 색인하는 TPR-트리[4]와 비교하였다.

성능평가에 사용된 파라미터들은 표 1과 같다. 먼저, 데이터의 수를 변경시키며 최대 100,000개의 데이터를 색인 구조에 삽입하는 시간을 측정하였고 100,000개의 데이터가 색인된 색인 구조에 최대 10,000개의 데이터가 갱신되는데 소요되는 시간을 측정하였다. 또한, 미래 위치 검색 처리 성능을 평가하기 위해서 100,000개의 데이터가 색인된 색인 구조에 5~30분 후 범위 질의 1000개를 처리하는데 소요되는 시간과 20분 후의 범위 질의에 대해 질의 영역의 크기를 변경시키면서 미래 위치 검색을 처리하는데 소요되는 시간을 측정하였다.

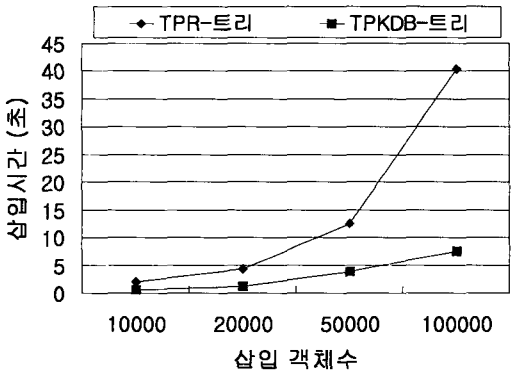
〈표 1〉 성능평가 파라미터

항 목	값
삽입 이동 객체 수 [개]	10,000 ~ 100,000
갱신 객체 수 [개]	500 ~ 10,000
질의 시간 간격 [분]	5 ~ 30
질의 영역 크기 [전체영역 %]	0.1 ~ 5

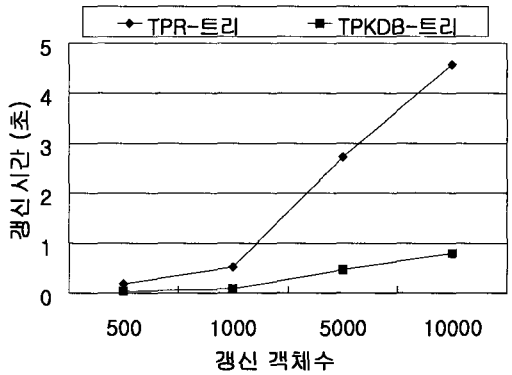
4.2 성능 평가

제안하는 TPKDB-트리는 계속적인 위치 변경이 발생하는 이동 객체에 대한 갱신 비용을 최소화하고 현재 및 미래 위치에 대한 검색 성능을 향상시키기 위한 색인 구조이다. 제안하는 색인 구조의 우수성을 입증하기 위해 기존에 제안된 TPR-트리와 제안하는 TPKDB-트리에 대한 색인 구성 시간 및 검색 시간에 대한 비교를 수행한다. 색인 구성 시간에 대한 비교는 삽입 시간과 갱신 시간 측면에서 비교를 수행하고 검색 시간에 대한 비교는 현재와 미래 위치에 대한 범위 질의를 수행한다.

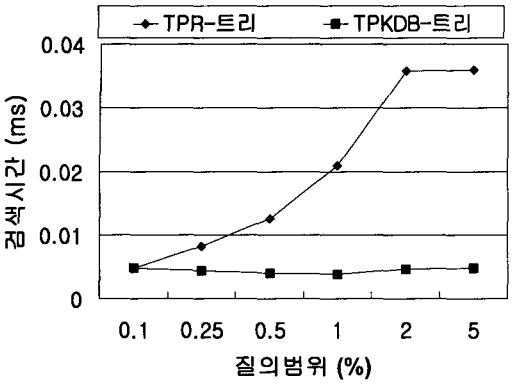
그림 11과 그림 12는 제안하는 TPKDB-트리와 TPR-트리에 대한 삽입 시간 및 갱신 시간을 비교한 결과이다. 그림 11은 데이터의 개수를 10,000~100,000개까지 변경시키면서 삽입 성능을 실험한 결과이다. 또한 그림 12는 100,000개의 데이터를 색인하고 있는 각 색인 구조에 데이터의 개수를 500~10,000개까지 변경시키면서 갱신 성능을 실험한 결과이다. 성능평가 결과 TPKDB-트리가 TPR-트리 보다 모든 경우에서 우수한 성능을 나타내었다. 삽입되는 객체의 수가 증가할수록 TPKDB-트리가 우수한 성능을 나타낸다. 성능 평가를 수행한 결과 삽입 성능은 기존의 TPR-트리에 비해 300%~550%의 성능 향상되었으며 갱신 성능은 600%~700%의 성능 향상을 나타낸다. TPR-트리는 데이터 분할 방법을 사용하기 때문에 데이터의 계속되는 위치 이동으로 인해 색인의 변경이 빈번히 발생하기 때문에 많은 시간이 요



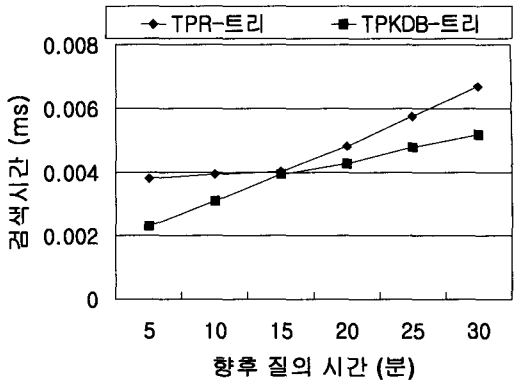
〈그림 11〉 TPKDB-트리와 TPR-트리의 삽입 시간



〈그림 12〉 TPKDB와 TPR 트리의 갱신 시간



〈그림 13〉 TPKDB-트리와 TPR-트리의 검색 시간



〈그림 14〉 미래 위치 검색 시간

구된다. TPKDB-트리는 공간 분할 방법을 사용하여 데이터를 색인하기 때문에 데이터 분할 방법을 사용하는 색인 구조의 문제점인 빈번한 색인 구조의 변경 비용을 줄일 수 있고 보조 색인 구조를 통한 빠른 객체의 검색을 지원함으로 갱신 비용을 효율적으로 줄일 수 있다.

그림 13과 그림 14는 TPR-트리와 제안하는 TPKDB-트리에 대한 현재와 미래 위치에 대한 범위 질의에 대한 성능을 평가를 수행한 결과이다. 그림 13은 범위 질의 영역의 크기를 전체 영역에 대해 0.1~5%까지 변경시키면서 1000개의 질의를 처리하는 검색 성능을 실험한 결과이다. 또한 그림 14는 미래 위치 검색을 위해 범위 질의 시간을 5~30분까지 변경시키면서 1000개의 질의를 처리하는 검색 성능을 실험한 결과이다. 성능평가 결과 TPKDB-트리가 TPR-트리보다 모든 경우에서 우수한 성능을 나타내었으며 삽입되는 객체의 수가 증가할수록 TPKDB-트리가 우수한 성능을 나타내었다. 성능평가를 수행한 결과 현재 위치에 대한 범위 질의는 100%~800%의 성능 향상을 나타내고 미래 위치 검색 성능은 100%~200%의 성능 향상을 나타낸다. TPR-트리는 데이터를 색인하는 MBR 또는 미래 위치 검색 시 확장되는 MBR의 많은 겹침과 색인의 빈번한 변경으로 전체적인 색인의 검색 성능이 저하되고, TPKDB-트리는 데이터 색인 시 영역 간에 겹침이 발생하지 않는 장점과 미래 위치 검색 시 불필요한 노드의 확장을 줄이고자 노드 내에 포함되어 있는 이동 객체의 변화를 시간에 대한 파라미터로 유지함으로서 검색 성능을 향상시킬 수 있다.

5. 결 론

본 논문에서는 효율적으로 이동 객체를 관리하고 미래 위치를 검색할 수 있는 색인 구조를 제안하였다. 제안된 TPKDB-트리는 이동 객체의 갱신 비용을 줄이기 위해 성능이 개선된 KDB-트리와 보조 색인 구조의 혼합형 색인 구조이다. 제안한 색인 구조에서는 빠른 이동 객체의 미래 위치 검색을 위해 노드와 노드 안에 포함되어 있는 이동 객체의 관계를 시간에 대한 파라미터로 유지하였다. 또한, 색인 구조의 공간활용도를 향상시키기 위해서 새로운 갱신 및 분할 기법을 제안하였고 성능 평가를 통해 기존 색인 구조에 비해 갱신과 검색 성능이 향상되었음을 보였다. 향후 연구 방향으로 다양한 실험을 수행하고 다양한 유형의 검색 기법들에 대한 연구들을 진행할 예정이다.

참 고 문 헌

[1] Rui Ding, Xiaofeng Meng and Yun Bai, "Efficient index update for moving objects with

future trajectories", Proc. Eighth International Conference on Database Systems for Advanced Applications, pp.26-28, 2003.

[2] A. Guttman, "R-trees : A dynamic index structure for spatial searching", Proc. ACM SIGMOD, pp.47-57, 1984.

[3] S. Prabhakar, Yuni Xia, Kalashnikov, D.V., W.G. Aref and S.E. Hambrusch, "Query indexing and velocity constrained indexing : scalable techniques for continuous queries on moving objects", IEEE Transactions on Computer, Vol.51, pp.1124-1140, 2002.

[4] S. Simonas, Christian S. Jensen, Scott T. Leutenegger and Mario A. Lopez, "Indexing the Positions of Continuously Moving Objects", Proc. ACM SIGMOD, pp.331-342, 2000.

[5] Dongseop Kwon, Sangjun Lee and Sukho Lee, "Indexing the Current Positions of Moving Objects Using the Lazy Update R-tree", Proc. Third International Conference on Mobile Data Management, pp.113-120, 2002.

[6] 전봉기, 임덕성, 홍봉희, "이동체 데이터베이스를 위한 색인 기법", 데이터베이스 연구회지, Vol18, No4, 2002.

[7] Yuni Xia and Sunil Prabhakar, "Q+Rtree: Efficient Indexing for Moving Object Database", Proc. DASFAA, pp.175-182, 2003.

[8] 이창헌, 임덕성, 홍봉희, "KDB-Tree를 사용한 이동체 색인의 동적 변경", KDBC2002 학술발표논문집, Vol.18, No.2, pp. 117, 2002.

[9] Ratko Orlandic and Byunggu Yu, "Implementing KDB-Trees to support High-Dimensional Data", Proc. IDEAS, pp.58-67, 2001.

[10] A. Henrich, H.-S. Six and P. Widmayer, "The LSD Tree : Spatial Access to Multidimensional Point and Non-point Objects", Proc. VLDB, pp.45-53, 1989.

[11] J. M. Hellerstein, J. F. Naughton, and A. Pfeffer, "Generalized Search Trees for Database Systems", Proc. VLDB, pp. 562-573, 1995.

◐ 저 자 소 개 ◑

서 동 민

2002년 : 충북대학교 정보통신공학과(공학사)
2004년 : 충북대학교 정보통신공학과(공학석사)
2004년~현재 : 충북대학교 정보통신공학과 박사과정
관심분야 : 데이터베이스 시스템, 에이전트 시스템, XML, 이동 객체 데이터베이스, 시공간 색인 구조 등
E-mail : dmseo@netdb.chungbuk.ac.kr

복 경 수

1998년 : 충북대학교 수학과(이학사)
2000년 : 충북대학교 정보통신공학과(공학석사)
2000년~현재 : 충북대학교 정보통신공학과 박사과정
관심분야 : 내용기반 멀티미디어 검색, 저장 시스템, 고차원 색인 구조, 이동 객체 데이터베이스, 시공간 색인 구조 등
E-mail : ksbok@netdb.chungbuk.ac.kr

유 재 수

1989년 : 전북대학교 컴퓨터공학과(공학사)
1991년 : 한국과학기술원 전산학과(공학석사)
1995년 : 한국과학기술원 전산학과(공학박사)
1995년~1996년 8월: 목포대학교 전산통계학과 전임강사
1996년 8월~현재 : 충북대학교 전기전자컴퓨터공학부 부교수
관심분야 : 데이터베이스 시스템, 정보검색, 멀티미디어 데이터베이스, 분산 객체 컴퓨팅 등
E-mail : yjs@cbucc.chungbuk.ac.kr

이 병 엽

1991년 : 한국과학기술원 전산학과(공학사)
1993년 : 한국과학기술원 전산학과(공학석사)
1997년 : 한국과학기술원 경영정보공학(공학박사)
1997년~2003년 : 대우정보시스템 CRM사업팀 차장
2003년~현재 : 배재대학교 전자상거래학부 전임강사
관심분야 : 데이터마이닝, XML, 인공지능, 멀티미디어 데이터베이스, 분산 객체 컴퓨팅 분야 등
E-mail : bylee@pcu.ac.kr